Assignment 3

CS-GY 6033 INET Fall 2024

Due date:
Nov. 4th 2024, at 11:55pm on Gradescope.

Instructions:
Justify all your answers using techniques from class. Your solution is to be written out (or typed) and submitted online through Gradescope before the deadline.

Question 1: Binary Search trees

.

(a) 4 points. Given the input$5, 1, 4, 2, 7, 9, 3, 6$, count the number of comparisons carried out when a BST is built by inserting the keys in their given order. Recall that in class, we showed that the number of comparisons carried out by the BST building process is the *same* as the number of comparisons carried out by Quicksort. Your job is to show the execution of quicksort (and which pivots) corresponds to the BST that you built. Be sure to illustrate that both procedures use the exact same number of comparisons.

(b) 3 points Suppose we build a BST on n elements, by first permuting the input sequence, and then inserting the elements in random order. After $n-1$ elements are inserted, we have one element left to insert. How many possible positions are there for the last element to be positioned in the tree? Explain your answer.

(c) 12 points Let T be the root node of a non-empty Binary Search tree. Consider the following algorithms:

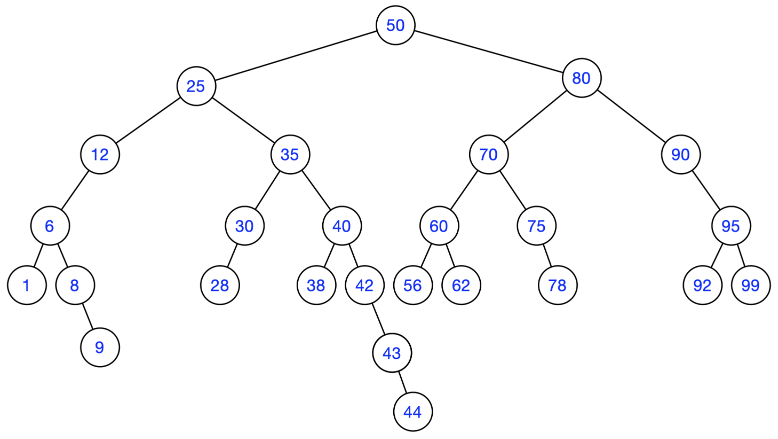
```
PrintTree1(T)
    if T ≠ NIL
        if T.left ≠ NIL
            print T.key
        PrintTree1(T.left)
        PrintTree1(T.right)
```

```
PrintTree2(T)
    if T ≠ NIL
        if T.left ≠ NIL
            print T.key
            PrintTree2(T.left)
            PrintTree2(T.right)
```

```
PrintTree3(T)
    if T = NIL
        return 0
    else
        $h$ = PrintTree3(T.left) + 1
        if $h > 4$
            print T.key
        return $h$
```

```
PrintTree4(T)
    if T = NIL
        return 0
    else
        h = PrintTree4(T.left) + 1
        print h
        return $h$
```

Describe in English the printed output of each of the above algorithms. Execute each of these algorithms on the following tree:



(d) 4 points Let T_1 and T_2 be two red-black trees. Each tree has n nodes. Explain how to create *one* red-black tree out of the merged set of elements, in time $O(n)$.

Question 2: Tree traversals

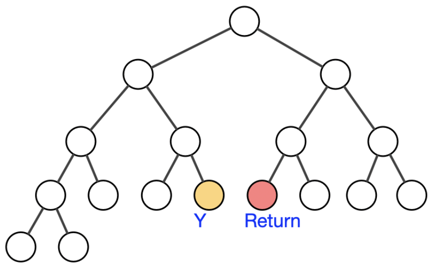
.

(a) 6 points Let T be a reference to the root node of a binary search tree. Write the pseudo-code for a **recursive** algorithm called TrimTree(T) that deletes all nodes that have **only one child**. Your algorithm must return a reference to the resulting tree. You must justify the runtime of $O(n)$. You are **not** allowed to call any algorithms from class.

(b) 6 points Let A be an array of n elements, indexed from 1 to n, which represent the post-order traversal of a BST. Your job is to write a procedure that re-creates the tree from this input. You must provide the pseudo-code for a **recursive** algorithm, called RecreateBST(A,s,f) which takes as input the array A and the indices that define the array elements to be used. The initial call will be to RecreateBST$(A, 1, n)$.
**You may use x = New Tree node() when you want to create a new node object*

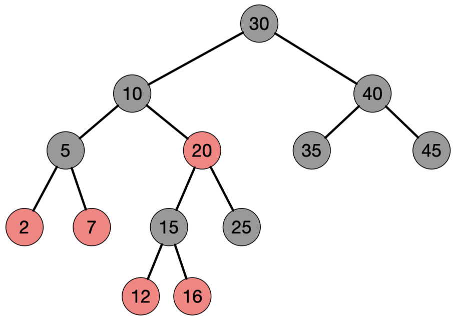
(c) 6 points Write the pseudo-code for an algorithm that takes in a reference to the root of a binary search tree, and prints out nodes at the *deepest* level. For example, if tree T has height 3, the algorithm should print out all nodes at depth 3. Justify the runtime of $O(n)$. You may find it helpful to call a recursive procedure within your main algorithm.

(d) 6 points
Let T be the root node of a **complete** binary search tree. Recall that complete binary trees are such that every level is *full* except perhaps the last level, which is 'filled-in' from left to right. Let y be a reference to a node in the tree that is *not* in the last level. The goal is to return a reference to the node that would follow y in 'level-order' traversal. Your job is to write the pseudo-code for an algorithm called MyNeighbour(T, y) which returns a reference to the node which follows y in 'level-order' traversal. You can assume that y is not in the last level, and then y is not the right-most node in its level. Justify the runtime of $O(h)$.



Question 3: Red-black trees

(a) 6 points
The drawing to the right epresents a valid red-black tree, augmented with subtree sizes. Show how to insert the new node 17. You must show both the initial insertion, and any changes made by RB-repair. You must also show the updates to any attributes of the tree nodes. Explain briefly why the black height of the tree does not change after this insert. After 17 is inserted, is there another node that could be inserted which would cause the black-heigh to increase?



(b) 10 points For each of the following statements, decide if the statement is true for false, and briefly (one sentence!) justify your answer.

1. A red-black tree has black height 4. What is the maximum number of nodes possible in the tree?
2. A red-black tree has 45 nodes. What is the smallest possible black height for such a tree?
3. A red-black tree has 45 nodes. What is the largest possible tree height?
4. A red-black tree has 45 nodes. Is it possible that all nodes in the tree are colored black?
5. A red-black tree has 7 nodes and height 2. Is there only one possible coloring for such a tree?

(c) 4 points

Suppose that there are n nodes in a RBT. What is the maximum black-height of the tree? (express your answer in terms of n). What is the maximum *height* of tree ? (express your answer in terms of n)? Use this result to determine if it is possible to have a red-black tree on 100 nodes of height 15. If so, draw the tree, if not, explain why not.

(d) 3 points. Draw an example of a RBT on 16 nodes with black height 3 that has two different possible colorings. Show the different possible colorings.

(e) 4 points Explain why it takes $O(n \log n)$ time to build a RBT. Next, suppose we have already built a BST. Explain how to *convert* the BST to a RBT in $O(n)$ time. How you do the conversion is up to you (you don't have to maintain the original shape in any way).

Question 4: Warm up to Augmented trees

(a) 6 points Let T be the root node of a Binary Search tree. We wish to augment each node x of the binary search with an attribute $x.diff$, which represents the *difference* between the longest path down the left subtree and the longest path down the right subtree. For example, if $x.diff = 3$, then the longest path down the left subtree is three more than the longest path down the right subtree.Write the pseudo-code for a **recursive** algorithm called SetDiff(T) that sets of the value of $x.diff$ for each node in the tree. To simplify your procedure, it may be useful to have it return a value, which you can use recursively. Justify that your procedure runs in $O(n)$.

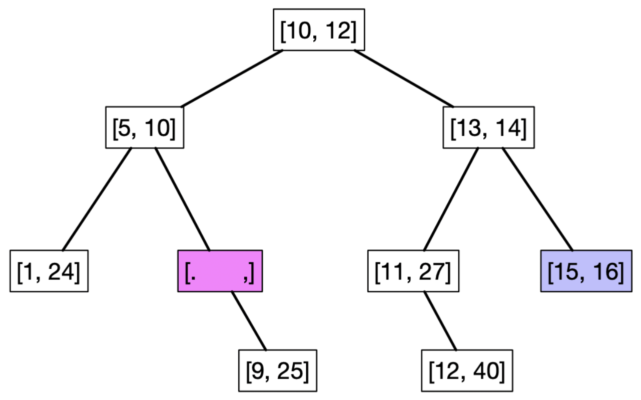
(b) 6 points

Let T be a reference to the root of a BST augmented with $x.diff$ as defined above. Your job is to update the Tree-Insert procedure so that it updates the values of $x.diff$ throughout the tree, when new node x is inserted. Justify why the runtime of Tree-insert is still $O(h)$.

(c) 4 points An interval tree is partially drawn below. Suppose we carry out the INTERVAL-SEARCH(i) algorithm from class.

1. Determine how to fill in the PINK node in such a way that the interval search algorithm will return the BLUE, [15, 16]. node. You must provide an input interval i that causes the return value, or explain why no such PINK node is possible.

2. Fill in the PINK node so that the interval search algorithm returns the pink node, when we search for an overlap with $i = [25, 26]$ or explain why it is not possible.



.**(d) 4 points** Rewrite the interval search algorithm so that it prints out all intervals that intersect with the input interval i. Call your procedure PrintAll(T, i), where T is a reference to the root node of an interval tree, and i is an interval object. The result is that ALL intervals in the tree that intersect with i are returned.

(e) 2 points How can you implement an interval tree so that deletions and insertions are carried out in time $O(\log n)$?

Question 5: Augmented BSTs

(a) 4 points Let T be an interval tree that stores information on projects running in a company . Each tree node x contains the following attributes: **x.name** (the name of the project), **x.start** (the start time of the project), **x.end** (the end time), **x.budget** (the project budget), **x.max** (the usual max attribute used in interval trees), **x.size** (the size of the subtree rooted at x) and **x.btotal**, (the total of all budgets for projects in the subtree rooted at x.)

Write the pseudo-code for a recursive algorithm called EarlyBudget(T, s) which returns the total budget for *all* projects in the tree which start before time s. You must justify the runtime of $O(h)$ of your algorithm, where h is the height of your tree.

(b) 4 points Refer to the above interval tree. Write the pseudo-code for a number of projects that start *after* project k finishes. Cal your algorithm PorjectsAfter(T, k) You must justify the runtime of $O(h)$ of your algorithm, where h is the height of your tree.

(c) 4 points Refer to the above interval tree. Write the pseudo-code for a recursive algorithm that the total budget of all projects that start *after* project k starts. Assume that k is a reference to a project in the tree. Call your procedure PostBudget(T, k). Justify the runtime of $O(h)$.

(d) 4 points Write the pseudo-code for the algorithm that prints out the number of projects that start between time a and time b. Call your procedureIntervalTotal(T, a, b). Justify the runtime of $O(h)$.

(e) 6 points Update the tree-interval procedure from class so that it takes in a new project k and correctly inserts it into this Interval tree. All attributes must be correctly updated. Call your procedure ProjectInsert(T, k). Justify the runtime of $O(k)$.

BONUS: 10 points (grade is either 0, 5, or 10. Write a procedure for an algorithm that determines the **maximum** number of projects running at any one time.